An introduction to DATA COMMUNICATION

by P. J. Cadman, G4JCP*

First licensed as G8HHK in 1973 after some three years as an swl, and as G4JCP since the end of 1979, the author is a graduate of Aston University. After spending a year with a company making electric arc furnaces he returned to Aston to do research on data transmission. During this period he became interested in the use of microprocessors, particularly their use in communications systems and associated equipment, and he still pursues this interest, but only as a hobby at the moment. For the last two years he has been working on the design of microprocessor-based equipment to replace traditional hard-wired logic units at Static Systems Group, in Wombourne, Wolverhampton. His current interests are amateur packet radio, computers, and getting away from the local QRM.

PACKET RADIO is a term which seems to be appearing regularly in connection with amateur radio and data transmission. As packet radio has its beginnings in the world of computer networks and high-speed data communication, the average amateur may be wondering what it is all about. The purpose of this article is to describe what is generally meant by the term "packet radio" and how it forms a logical progression from the more-widely-known forms of serial data communication.

Most data communication between computers and their input/output devices such as vdus takes place serially using an asychronous format. Here each character is transmitted one bit at a time. Fig 1 shows the bit pattern that represents the letter R sent asychronously in Ascii (American standard code for information interchange). The speed at which characters are sent is usually expressed in bauds. The baud rate is equal to the number of discrete changes in the signal that can take place each second. When the signal can only assume two states, as in Fig 1, the baud rate equates exactly to the bit rate in bits/second. If the signal can assume more than two discrete states then the bit rate can, in theory, be higher than the baud rate as now more than one bit of information can be represented by each state. There are preferred bit rates for asynchronous communications; namely 50, 110, 150, 300, 600, 1,200, 2,400, 4,800, 9,600 and 19,200 bits/s, but in principle there is no restriction placed on the bit rate providing, of course, both transmitter and receiver are in agreement.

Asynchronous transmission

Surrounding each character are extra bits which are necessary for asynchronous communication; these are called start and stop bits. There may also be an additional bit known as a parity bit. This bit is normally generated automatically by the transmitter and subsequently removed automatically by the receiver, and is used to check for errors in the received character. The logical state of the parity bit is chosen to make the total number of "1"s in the character plus parity bit either an odd number (odd parity), or an even number (even parity). When the transmitted character is received the state of the parity bit is once again calculated and compared with the received parity bit. If they are different then it is assumed that one bit in the character has changed during transmission. Of course the received parity bit itself may be in error but there is no way of deducing this at the receiver. Strictly, a single parity bit will detect any odd number of errors in a received character but not an even number of errors. Therefore a single parity bit should not be relied on when there is a significant probability of multiple errors occurring in a single character.

The start and stop bits added to the beginning and end of the character are for timing purposes. They, like the parity bit, will be added and removed automatically by the uarts at each end of the communications channel. A uart (universal asynchronous receiver/transmitter) is the dedicated communications device which performs all the formatting, timing and checking that serial data transmission requires. Each transmitted character begins with a single, logic zero, start bit and ends with either 1, 1·5 or 2, logic one, stop bits. The junction of the stop bit(s) of one character and the start bit of the following character form a one-to-zero transition which is present at the beginning of all transmitted characters. The timing circuits in the receiving uart synchronize to this edge in the following way.

In general a uart's synchronization circuitry is driven by a clock running at 16 times the bit rate. Initially the received signal is sampled each clock period, and as long as it remains in the "1" state (which is the "rest" state of the channel) the uart does nothing. As soon as the "0" state is detected the uart will wait another 7 or 7·5 clock periods and sample the signal again. If it is still in the "0" state the uart assumes it has found a valid start bit. The character is now shifted into the uart's receiving shift register one bit every 16 clock periods. The half-bit (eight clock periods) offset introduced ensures that the received signal is sampled (within a few per cent) at the centre of each bit period. When the entire character has been received it is checked for parity and for the presence of the correct number of stop bits. In the meantime the synchronization circuitry will have begun searching for the next start bit halfway through the first stop bit.

The great advantage of asynchronous communication is that only a single communications channel and a local 16-times clock are required. In addition, the hardware for both microprocessor and stand-alone use is very inexpensive, making it very popular for both commercial and amateur data communications in all but the more stringent applications. However, asynchronous transmission does have its limitations, one of which is speed. The 16-times clock means that the highest usable bit rate is only one-sixteenth of the maximum clocking frequency that the uart can cope with. Also, the addition of two or more extra bits per transmitted character reduces the efficiency of the communications channel. While this may not be a problem where the channel is a direct wired line, limited bandwidth

*21 Scotts Green Close, Scotts Green, Dudley, W Midlands DY1 2DX.

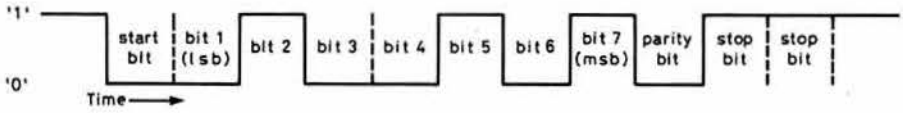

Fig 1. The letter "R" in Ascii with odd parity and two stop bits

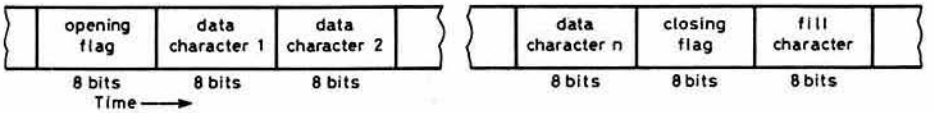

Fig 2. Synchronous transmission format using eight-bit flag and data characters

channels such as telephone circuits and narrow-band radio channels can have their potential data-carrying capacity reduced by 25 per cent or more. As has already been said, the error detection capability of asynchronous transmission is not good, making it totally unsuitable for communicating data which contains little or no natural redundancy. To give an example, plain text has natural redundancy and can, therefore, contain errors on reception which will be obvious (and correctable) by anyone reading the text. Other material may contain little or no natural redundancy. Computer programs and numeric tables are two examples that fall into this latter category. There is one further problem; asynchronous transmission is not suitable, as it stands, for multiple users who share a common communications channel. It has no inherent method of addressing multiple users, which means that some additional supervision of the received data is necessary.

Synchronous communication

Synchronous communication goes some way to overcoming the limitations of asynchronous communication. The most notable difference is that no start and stop bits are transmitted, the synchronization being provided by a clock transmitted with, but separate from, the data. This immediately reveals the major disadvantage of this type of transmission, in that two channels are now necessary. Over moderate distances this is usually acceptable and at least the usrt (universal synchronous receiver/transmitter) can transmit data at its maximum clocking frequency. Readers may have come across the term usart, which describes a programmable device which can handle both synchronous and asynchronous communications, though clearly not at the same time. The separate clock synchronizes the receiver on a bit-by-bit basis but not on a character basis. Indeed, all forms of synchronous receiver suffer from the inability to synchronize to a message if the starting sequence has been missed. This is due to the nature of the character/message synchronization used, as illustrated in Fig 2.

The beginning and end of the message or, more correctly, the frame (a message may consist of several frames), are marked by a special pattern of bits known as a flag. This flag may also be continuously transmitted when there is no data available, thereby not allowing the channel to remain in one state for any length of time. This is useful from a system integrity standpoint, as now any break in the channel will be detected immediately whether or not a frame is being transmitted. The absence of start and stop bits means that now almost all the channel capacity can be used for data, the only overhead being the flags surrounding the frame. However, there is still no method of checking the integrity of the received data save for the possibility of a single parity check bit, appended to each character, as in the case of asynchronous transmission. Any additional error checking would require external hardware (or software) to encode and decode the frame. Synchronous communication in itself is still not suitable for multi-user situations, as these would require more complex operations to be performed by the usrts. Such devices are now readily available and are known as high-level data link controllers.

Data link controllers

Synchronous communication techniques form the basis of networking systems, but when multiple users share a single data link a considerable amount of software and processor time is needed to manage the system. In addition, different groups of people are likely to end up with incompatible systems which can cause great problems when previously isolated groups wish to communicate. Fortunately any problems have, in the main, been resolved by a few large organizations (standards authorities and large computer companies) who have developed data-link protocols to handle multiple users and high-integrity communications. Most large integrated circuit manufacturers have produced high-level data link controllers that implement these protocols in hardware on a single chip. They are powerful enough to off-load from a microprocessor much of the work involved in managing high-level data link communications. These controllers are not stand-alone devices as are some uarts; they require a microprocessor and support circuits to operate and most make provision to interface to other microprocessor hardware (such as direct memory access controllers) to further automate data communications. It is the availability of these controllers at relatively low cost (about £11 to £40) and suitable microprocessor systems that makes amateur packet radio possible.

High-level data link protocols do vary one from another but all share a similar structure, so a brief description of a typical protocol will suffice to show what is involved. Fig 3 shows the make-up of a typical data frame; notice that now individual bits are no longer significant in themselves, but it is the pattern formed by successive bits which is important. The frame again begins with a flag but now it is fixed at "01111110". This pattern is most important for reasons which will become clear. Going back for a moment to basic synchronous communications, the flag which marks the beginning (and end) of a frame can be almost any bit pattern, take for example '01101010'. If a usrt is now instructed to look for this flag while data is in the process of being transmitted, a data character or combination of two data characters may accidentally form this pattern. The usrt will mistake it for a genuine flag and so incorrectly indicate the start of a frame. An example of how this may occur is shown in Fig 4. To prevent this happening in high-level data link communications, the pattern "01111110" is unique, hence it is not allowed to appear accidentally in the data stream. The link controller constantly monitors the transmitted data so that, following a run of five consecutive "1"s, a "0" is automatically inserted into the frame. The receiving controller also monitors the data, automatically removing a "0" which follows five consecutive "1"s.

Following the opening flag is an address field, this is usually the address of the station for which the data is intended, although it may also contain the address of the sender. This field is made up of one or more eight-bit characters. The automatic zero insertion just mentioned, also known as bit-stuffing, is enabled immediately following the opening flag so its operation may affect the address field. Because it occurs transparently its operation and effect on the transmitted data stream is ignored. Hence the length of the eight-bit characters of the address field is that before any bit-stuffing takes place. After the address field comes the control field, consisting of one or two eight-bit characters. These are, to a large extent, user definable and will typically indicate control functions relating to the system hardware and to the data link's operation. The data field itself follows the control field. It may contain any number of bits, including zero bits, and need not be a multiple of a fixed-length character although this would normally be the case. In amateur applications and personal computer local area networks the data field will typically consist of eight-bit characters simply because this matches the data word length used by eight- and sixteen-bit microprocessors. Immediately preceding the closing flag, again "01111110", there is a frame check sequence (fcs) of 16 bits generated by the transmitter from all the bits in the frame except for the flag bits. It can be viewed as the equivalent of the parity bit in asynchronous transmissions but is far more secure. An fcs is generated by the receiver from the received data bits, and once the closing flag is detected the receiver's fcs is checked

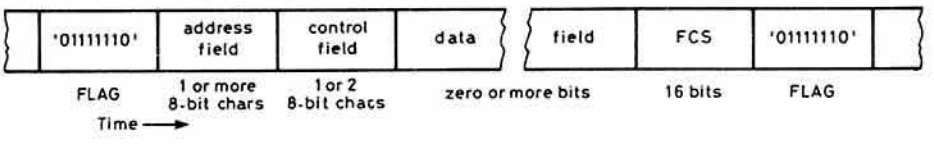

Fig 3. HDLC frame

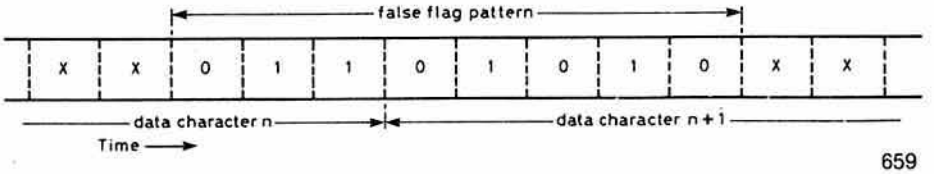

Fig 4. False flag recognition

against the expected result. If any number of bits were corrupted during transmission it would be extremely unlikely that the correct fcs would be formed. In addition to the opening/closing flag there are a few other bit patterns which have six or more consecutive "1"s, and they all have special meanings: for example, "01111111" means go-ahead, and seven (or more) "1"s within a frame means abort the current frame.

One thing not covered so far is: how does the clock necessary for synchronous communications get transmitted with the data over a single channel? To get around this problem at low bit rates at least, a "trick" known, perhaps a little incorrectly, as non-return-to-zero inverted (nrzi) encoding is used. A phase-locked-loop clock in the receiver running at 32 times the bit rate is synchronized to the incoming data bits by transitions in the data stream. To prevent long runs of "0"s (which can occur) causing the pll to become sufficiently out of step with the received data stream to cause errors, nrzi encoding causes a transition in the bit stream to represent a "0" and no transition, at the appropriate time, to represent a "1". Remember that runs of more than five consecutive "1"s are not normally permitted, so "0"s, and therefore transitions in the bit stream, are guaranteed to occur often enough to keep the pll in synchronism. Although this technique does severely limit the maximum bit rate, it is adequate for use over narrow bandwidth channels.

Terminal node controller

Amateur packet radio is closely related to the high-level data link protocol just described. It differs only in detail, such as extending the frame's address field to include the callsigns of both the sending and receiving stations. Although a personal computer can form the basis of a packet radio terminal node controller (tnc), the incorporation of a data link controller into the machine may cause problems and void any manufacturer's warranty. Most people will, therefore, prefer to buy or construct a dedicated tnc, a representative block diagram of which is shown in Fig 5. The choice of cpu/ram/eprom/uart and general purpose i/o port is largely up to the individual, any of the popular microprocessor families can be used with equal results. Indeed, the deciding factor is likely to be the individual's own experience with a particular family and if any packet radio software for a particular microprocessor is available. Where the choice is more open the microprocessor which has the best high-level language support should be used. Buying an off-the-shelf tnc naturally by-passes these problems. The choice of high-level data link controller, which in its single chip form will be referred to by the term hdlc, is not so open-ended. Not all hdlcs support nrzi encoding, current devices that do (to the author's knowledge) are the Intel 8273, the Western Digital WD1933/1935, the Texas TMS9903 and the SGS/ZILOG Z8530. All are established products except for the SGS device, which has only recently been introduced and hence it is somewhat of an unknown quantity. Notable among them is the TMS9903 which is specifically designed to interface with the TMS9900 series of microprocessors and is, consequently, not really suitable for use with other microprocessor families. It does have some advantages over the others, not least of which is its small size, 20-pin dil as opposed to 40-pin dil for the others, and it can also operate asynchronously. The SGS Z8530 appears, from its data sheet, to be a powerful device incorporating two independent full-duplex usarts in one 40-pin package. It supports Z80-style vectored interrupts, making it suitable for use in Z80/Z800, Z8000 and MC68000 series systems. Of the Intel 8273 and WD1933, the 8273 is slightly more expensive than the WD1933. The WD1933 is more difficult to interface than the 8273 and is less easy to obtain, which detracts somewhat from its lower cost. The uart in the tnc allows communication with either a vdu for stand-alone operation, or with a personal computer. The use of a pc greatly enhances the capabilities of the tnc, particularly if the pc has a printer and disk drives. Battery-backed-up ram (or eeprom) is almost a necessity if only a vdu is used, allowing system parameters (callsigns, bit rates etc) to be saved when the tnc is switched off. With a pc attached such parameters can be down-loaded from the pc on power-on and so non-volatile data memory within the tnc will not be necessary. The remaining part of the system is the modem. Short for modulator/demodulator, it is worthy of an entire article in its own right so discussion of it here will necessarily have to be very limited.

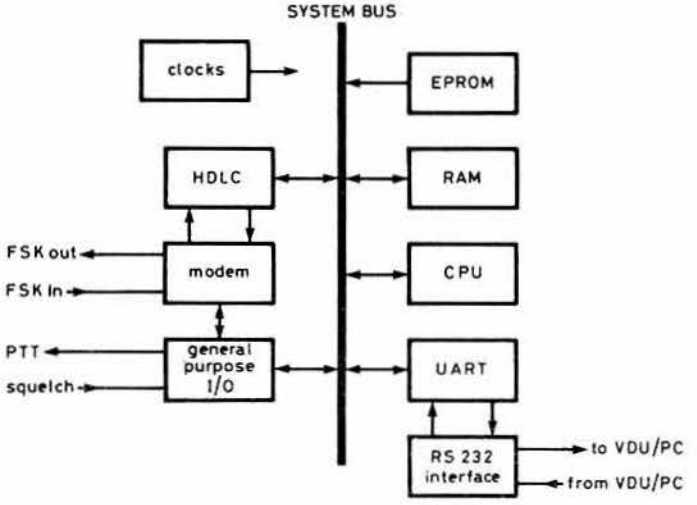

Fig 5. Block diagram of a typical terminal node controller

The modem is the device which converts the logic (voltage) levels output by the hdlc into signals suitable for transmission over the communications medium, in our case a radio channel. As most voice radio channels have some similarity to telephone channels, particularly in terms of frequency response and, to a lesser extent, noise, modified telephone data transmission techniques can be used quite successfully. The availability of telephone modems and their component parts at reasonable cost makes their use attractive. In the UK the adoption of 1,200Hz/2,400Hz fsk signalling has advantages, one of which is the compatibility this gives with both Uosat 1 and Uosat 2. The demodulator part of the modem is far more complex than the modulator, so most circuits that exist for demodulating the telemetry data from the Uosats could be used, saving time and hardware. As 1,200Hz/2,400Hz fsk is also part of the c.u.ts tape-recording standard used to record computer programs on cassette tape, again existing circuits can be used. The output of the modem is fed (after attenuation) into the microphone or phone-patch socket of a radio transceiver, while the headphone or phone-patch output will be suitable for feeding into the modem. However, a small amount of filtering may, in some cases, be beneficial at this point to compensate for any gross deviations the channel has from a flat overall frequency response and to restore a near linear phase response. An fm (or a.m.) transceiver is preferable to an ssb transceiver so as not to introduce any frequency/phase shifts in the received signals. SSB techniques which allow the re-inserted carrier to be correct in both frequency and phase may be of great assistance if data communication over hf channels becomes allowed within the terms of the amateur licence. Whatever method is chosen the better the modem then the more efficient will be the system, particularly at low snrs and in the presence of interference. Until recently data demodulators have always used some analogue circuitry and have required some sort of setting-up. With the introduction of signal processing microcomputers, such as the Texas TMS320, high performance demodulators using nothing more than a passive low-pass filter, an analogue to digital converter and a single-chip microcomputer may be within the financial reach of most amateurs fairly soon.

Amateur packet radio

Having decided on a common hardware specification, a common operating protocol needs to be adopted. The adoption of a common protocol is at least as important as the hardware specification, for unless all stations follow the same protocol satisfactory communication will not be possible. So just what is this protocol? All the hardware protocols already described achieve nothing more than the transmission and reception of single frames or packets of data. The response of both the transmitting and receiving stations following the transmission of a packet must cover the possibility of the packet being received with no detectable errors, being received with errors, or being completely lost. In the last two cases a retransmission will be necessary, so the receiver must be able to request such a retransmission and the transmitting station must recognize the request as such. If there is no response at all from the receiving station the transmitter should repeat the packet; how long the transmitter waits before doing so has to be decided in advance, as does the transmitter's action if several repeats fail to generate a reply. When there are more than two stations operating on one channel the operating protocol must now be capable of directing each station's access to the channel to prevent mutual interference. Clearly there must be one basic protocol to which all stations must adhere. However, it is quite permissible for some stations to have enhancements to the basic protocol as long as all stations behave similarly at the packet transmission level. A possible, much simplified, protocol might operate as follows—note that this assumes that all stations can achieve direct communication with each other so that no intermediate repeater stations are used:

Imagine station "A" wishes to pass several packets (which form a complete message) to station "B". "A" will first listen on the channel; if any other station is transmitting, "A" will wait for that station to finish. Once the channel becomes available "A" will transmit a short packet to "B", the control field of which will effectively ask "B" whether it is "on-line". "B" will respond with a similar packet which will (hopefully) say "yes". From now on "A" can send packets to "B", waiting between each one for an acknowledgement from "B". This form of operation is commonly known as automatic request repeat (arq). Once the entire message has been passed "A" will transmit a close down packet, thus freeing the channel for other stations to use. As with all shared channels there is always the possibility of two (or more) stations transmitting simultaneously. This can happen in this example following "A" 's close-down transmission if two other stations are waiting to pass messages. Both will transmit at the same time and so both packets will probably be lost. If both stations now wait the same length of time before trying again, the same thing might happen and this stalemate situation may continue for some time! To prevent this, following an unacknowledged packet the station should wait for a random length of time before it next transmits, thereby reducing the chance of further simultaneous transmissions. Should two-way communication be broken during the transmission of a message, both stations will close down and wait for some period of time before trying to resume communication; the packets that were successfully passed before contact was broken will not need to be repeated.

In a real situation the example protocol would only be usable over a single channel, and even then only if the overall channel usage was fairly low. The efficiency of all shared channels can fall dramatically as the channel occupancy approaches 100 per cent unless the operating protocol has specific provisions for high occupancy operation. The simple reason for this is because as the number of packets being transmitted increases then the probability of simultaneous transmissions increases (packets "collide"). Each collision reduces the effective channel capacity and so aggravates the situation. Ultimately the number of collisions may be so great that only a few packets are transmitted successfully, the net channel utilization dropping to a fraction of its optimum value. To cover most eventualities any protocol has to be comprehensive, resulting in some considerable programming of the tnc. To make the writing and maintenance of the protocol software quicker and easier, most of the programming should be done in a high-level language. The choice of language may be important as the program should, preferably, be portable from one microprocessor family to another. Suitable languages include compiled Basic (interpreted Basic will be too slow for all but very simple protocols), Fortran, BCPL, Modulo 2, C and Pascal. Compilers for both the latter languages are available at reasonable cost for most microprocessor families. Some parts of the program will have to be written in assembly code (for reasons of execution speed and to interface with specific hardware), and the ease with which this assembly code can be interfaced with the main program will also be important in the choice of high-level language. If the program is to be run on a dedicated tnc, the chosen compiler should be capable of producing "romable" code (code that is split up into at least two memory areas): one for the actual program, stored in rom; and one for the data the program uses and generates, which is located in ram. The program should also be complete in itself, making no calls to an external operating system.

Packet radio possibilities

While offering a method of communicating data reliably over local radio (and other voice) channels, packet radio has many other possibilities. Imagine a repeater station operating on two simplex frequencies: one would be the local channel, say 432MHz, and the other 1·3GHz. The antennas for the 1·3GHz channel would be high gain types directed at other nearby packet repeaters. A station wishing to pass a message to another station, possibly some hundreds of miles distant, would first pass the message to the repeater over the local channel. The repeater would, after ascertaining the area in which the destination station was situated, pass the message on to another repeater closer to that area. Once the message reached the destination station's local repeater it would be transmitted on the local frequency. Hopefully the message would then be passed successfully to the destination station. An acknowledgement of this last action would be passed back to the originating station. These repeaters could store any messages they could not pass on for, say, 12 to 24h. Although this is hardly what many people would term amateur radio, the benefits of such a system would be great to the increasing number of amateurs involved in various scientific studies and who, therefore, need to communicate data reliably to others. Maybe such repeater stations should operate on the microwave bands entirely, where the larger bandwidths available would allow high data rates and so make real-time multi-user packet communications possible.

Modification of a standard 144MHz Europa or Europa B to cover 70MHz

by M. GIBBINGS, G3FDW*

NO MAJOR ALTERATIONS to the Europa circuit are necessary for this modification, other than replacement of three variable capacitors in the receiver converter and replacement of self-supporting coils in the local oscillator chain and transmitter chain, plus the fitting of three or four new components. In addition to normal hand tools, a gdo for setting-up tuned circuits, covering 30-80MHz, is necessary. Before any modification is carried out, it is suggested that the unit is checked for satisfactory operation on 144MHz to ensure that all is well with the transverter to be modified.

Carry out the modifications step by step, in the order shown. First, the receiver converter is modified, followed by the local oscillator chains and the transmitter chain. On completion you will have an efficient and stable transverter which will give excellent results on the 70MHz band.

The following components and materials are required:

Three 3-30pF variable capacitors, large "piston" type similar to those fitted in receiver converter but with "red" or "pink" ceramic insulation.
Two 2-22pF variable capacitors. RS Components 125.654.
One 2·7pF ceramic capacitor.
One 4·7Ω, 1W resistor (non-inductive).
One 2·7kΩ 1/4 resistor.
One 0·25in ferrite bead, FX1115.
One 42·000MHz crystal HC18.
Wire, enamelled copper, 20swg and 22swg.
Wire, silver plated, 14swg and 18swg (for L1 and L11).
Wire, ptfe, insulated, tinned-copper 20swg (L12).

Details of coils to be replaced

L1: 11·5t close-wound 18swg silver plated wire, pull out to 1in length. Antenna tap at 3·5t.G1 of mosfet tap at 7·5t.
L2: 7·5t close-wound with 22swg enamelled-copper wire.
L3: 7·5t close-wound with 22swg enamelled-copper wire.
L5: 8t close-wound with 20swg enamelled-copper wire.
L6: 22t close-wound with 20swg enamelled-copper. Tap at 11t.
L7: 14t close-wound with 20swg enamelled-copper. Tap at 7t.
L8: 8t close-wound with 20swg enamelled-copper. Tap at 4t. Pull out to 1in long.
L9: 14t close-wound with 20swg enamelled-copper. Tap at 7t.
L11: 8t 14swg silvered-copper wire. Coil length overall to be 1·37in with 0·37in gap at 4t for ct and loading coil.
L12: 2t ptfe-insulated 20swg. Twist coil ends together over a length of 2·5in.

NOTE. All coils L1-L9 are wound on a 0·25in diameter drill or rod. L11 and L12 are wound on 0·75in diameter rod.

Make up all coils one by one and replace coils in turn as detailed, noting length of end connections and position relative to other components. All coils fit easily into space occupied by previous 144MHz coils.

Receiver modification

1. Remove antenna changeover relay and coaxial cable to series link variable capacitor. Disconnect capacitor to "output" circuit. Note connections carefully. Remove valves from transmitter.
2. Note how L1, L2 and L3 are mounted. Remove L1, L2 and L3.
3. Remove C1 and its back-to-back diodes, C2 and C3—these will need to be unsoldered and unbolted. Two of these capacitors are needed for the transmitter modification.
4. Fit new C1, C2 and C3 and resolder to top of printed circuit board.
5. Refit back-to-back diodes across C1.
6. Resolder mixer mosfet G1 to C3.
7. Fit a 0·25in ferrite bead to rf amplifier mosfet G2.
8. Make up and fit L3.
9. Make up and fit L2.
10. Make up and fit L1.
11. Solder 2·7kΩ resistor from G2 of mosfet mixer to ground (underside of printed circuit board).

NOTE. Refit L1-3 as noted in **2**. Mount L2 and L3 to have 0·19in gap between coils.

Local oscillator modification

1. Remove Q4 transistor.
2. Remove link from output of local oscillator chain to base of Q3 (via capacitor and four-turn coil). Leave capacitor and coil in circuit.
3. Remove red wire, of red/black twisted pair to mixer mosfet, from connection to o/p from original local oscillator chain. Connect this red wire to connection on printed circuit board from which link coil removed in **2**.

*"Glannaventa", Sturton-le-Steeple, Retford, Notts.